基于 Lasso, Ridge, ElasticNet, XGBoost 和 LightGBM 的集成模型分析

Kaggle 竞赛——House Prices

作者：20192131077 张帆　　20192131089 吴宇涛

目录

摘要

近年来机器学习成为一个热门学科，高等院校在计算机专业都有开设机器学习相关课程. 实践参加一次 kaggle 比赛并总结一份报告是快速入门机器学习的方法。本文通过总结机器学习 kaggle 竞赛《House Prices - Machine Learning from Disaster》，完成对房价的预测，以及阐述如何认识数据特征值，如何处理缺失值，数据可视化，使用 Lasso，Ridge，ElasticNet，XGBoost 和 LightGBM 的集成模型对房价进行预测.

关键词：Kaggle；House Prices；机器学习；Lasso；　Ridge；　ElasticNet；　XGBoost; LightGBM

1. 问题重述

1.1 问题背景

我们请购房者描述他们梦想中的房子时，他们可能不会从地下室天花板的高度或靠近铁路的程度去考虑. 但这场竞赛的数据集证明，与卧室数量或白色栅栏数量相比，那些因素对价格谈判的影响可能要大得多.

1.1.1 问题提出

在这个 Kaggle 竞赛中，我们需要用 79 个因素及变量来描述爱荷华州艾姆斯的住宅的各个方面，然后通过这些变量训练模型来预测房价.

2. 数据分析

2.1 数据获取

从 Kaggle 官网中下载三个.csv 表，使用 pandas 库读取 train.csv, test.csv, 从二维表可以看出我们有 79 列即 79 个特征. 我们需要通过数据处理来完成对缺失值的处理，并且确定哪些特征具有较强的预测能力，哪些是中等预测能力特征，哪些是弱预测能力特征.

```
In [3]: print(train)
```

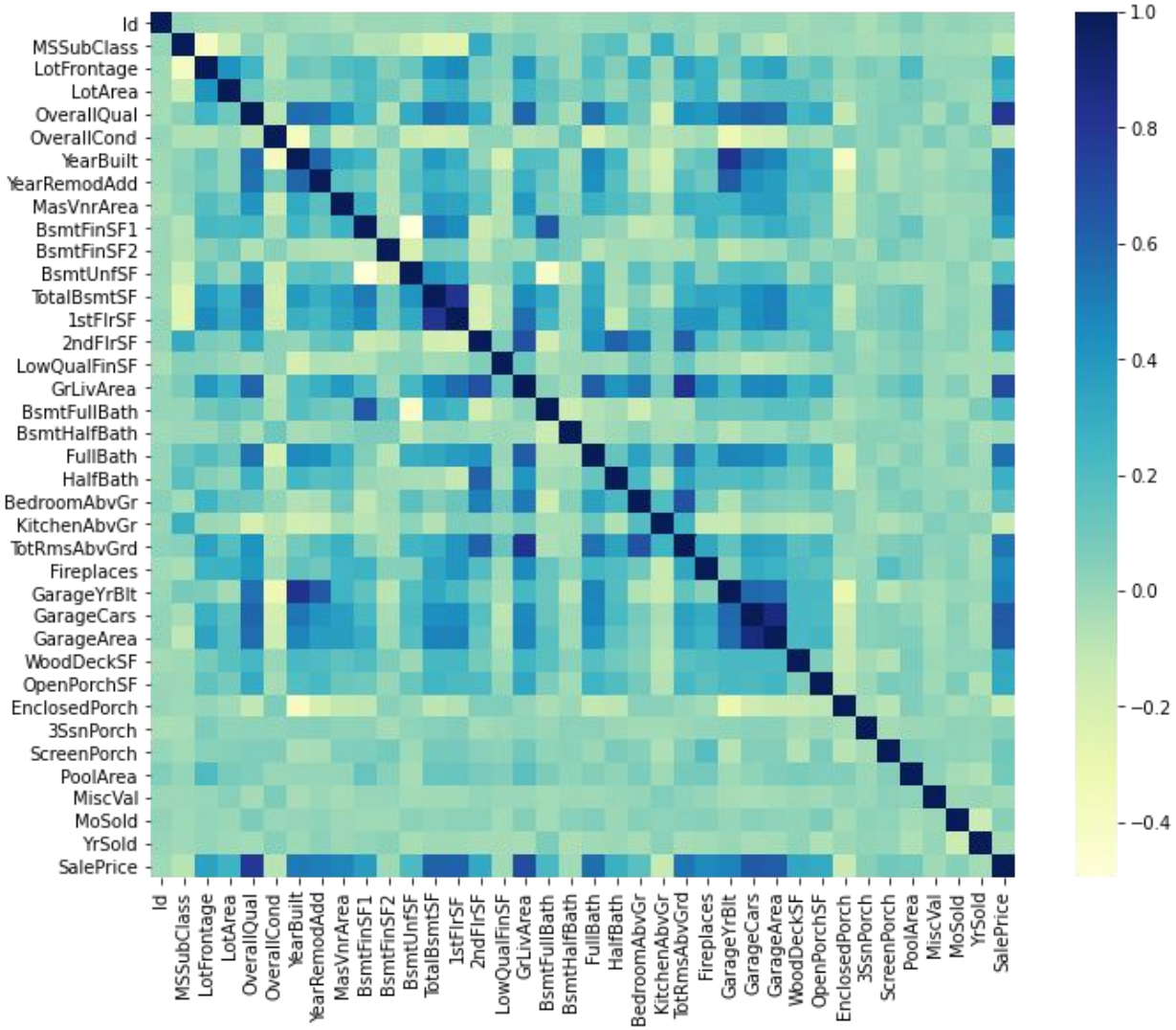
```
        Id  MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape  \
0        1          60       RL         65.0     8450   Pave   NaN      Reg
1        2          20       RL         80.0     9600   Pave   NaN      Reg
2        3          60       RL         68.0    11250   Pave   NaN      IR1
3        4          70       RL         60.0     9550   Pave   NaN      IR1
4        5          60       RL         84.0    14260   Pave   NaN      IR1
...    ...         ...      ...          ...      ...    ...   ...      ...
1455  1456          60       RL         62.0     7917   Pave   NaN      Reg
1456  1457          20       RL         85.0    13175   Pave   NaN      Reg
1457  1458          70       RL         66.0     9042   Pave   NaN      Reg
1458  1459          20       RL         68.0     9717   Pave   NaN      Reg
1459  1460          20       RL         75.0     9937   Pave   NaN      Reg

     LandContour Utilities  ... PoolArea PoolQC  Fence MiscFeature MiscVal  \
0            Lvl    AllPub  ...        0    NaN    NaN         NaN       0
1            Lvl    AllPub  ...        0    NaN    NaN         NaN       0
2            Lvl    AllPub  ...        0    NaN    NaN         NaN       0
3            Lvl    AllPub  ...        0    NaN    NaN         NaN       0
4            Lvl    AllPub  ...        0    NaN    NaN         NaN       0
...          ...       ...  ...      ...    ...    ...         ...     ...
1455         Lvl    AllPub  ...        0    NaN    NaN         NaN       0
1456         Lvl    AllPub  ...        0    NaN  MnPrv         NaN       0
1457         Lvl    AllPub  ...        0    NaN  GdPrv        Shed    2500
1458         Lvl    AllPub  ...        0    NaN    NaN         NaN       0
1459         Lvl    AllPub  ...        0    NaN    NaN         NaN       0

     MoSold YrSold  SaleType  SaleCondition  SalePrice
0         2   2008        WD         Normal     208500
1         5   2007        WD         Normal     181500
2         9   2008        WD         Normal     223500
3         2   2006        WD        Abnorml     140000
4        12   2008        WD         Normal     250000
...     ...    ...       ...            ...        ...
1455      8   2007        WD         Normal     175000
1456      2   2010        WD         Normal     210000
1457      5   2010        WD         Normal     266500
1458      4   2010        WD         Normal     142125
1459      6   2008        WD         Normal     147500

[1460 rows x 81 columns]
```

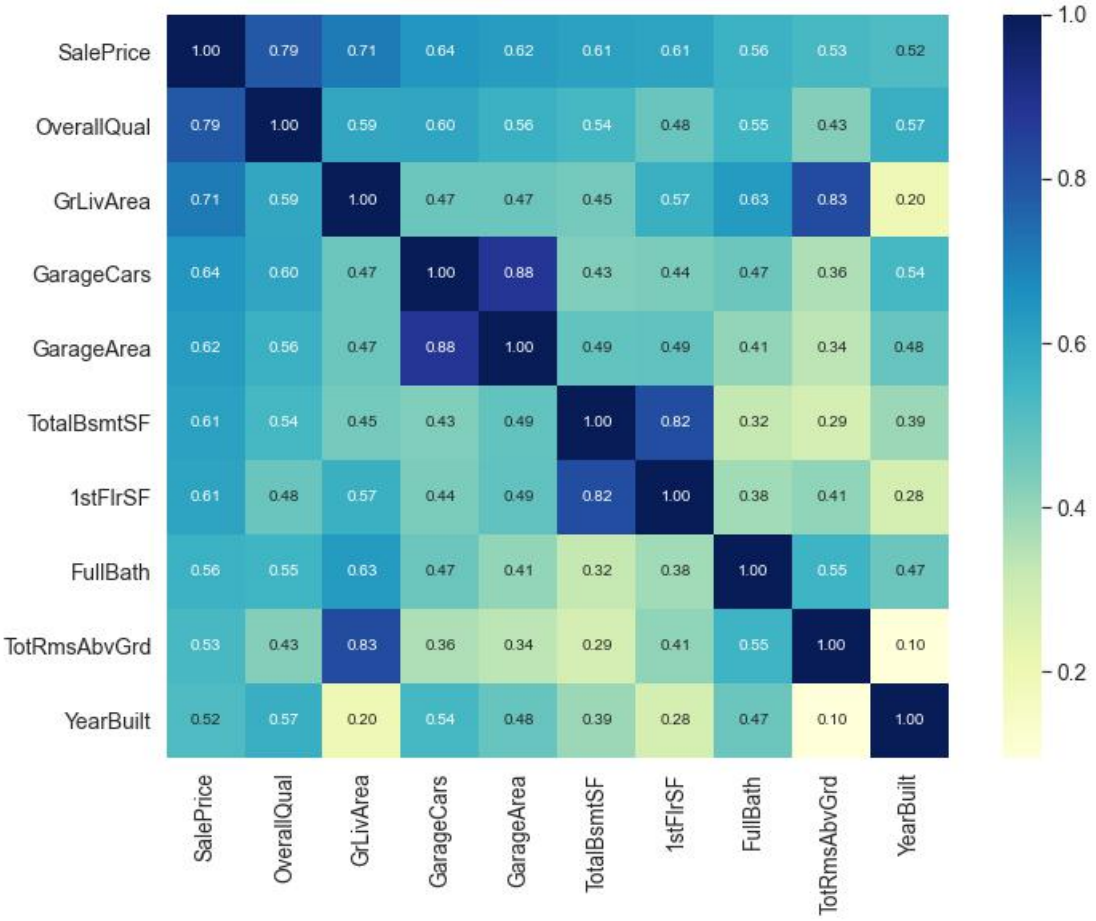
```
The train data size before dropping Id feature is : (1460, 81)
The test data size before dropping Id feature is : (1459, 80)

The train data size after dropping Id feature is : (1460, 80)
The test data size after dropping Id feature is : (1459, 79)
```

2.2 数据处理

2.2.1 特征总体分析

特征比较多，但我们可以通过热力图知晓哪些特征与价格之间相关性比较高

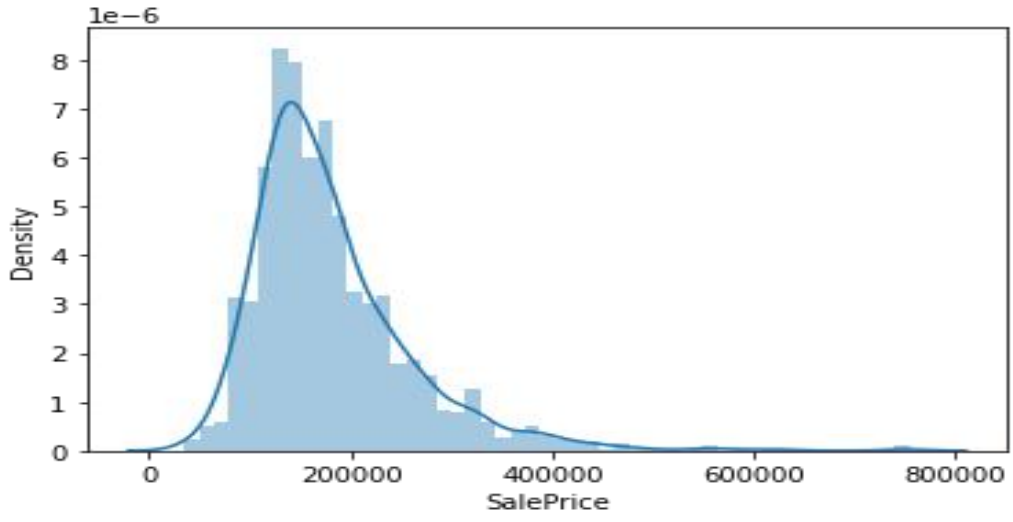


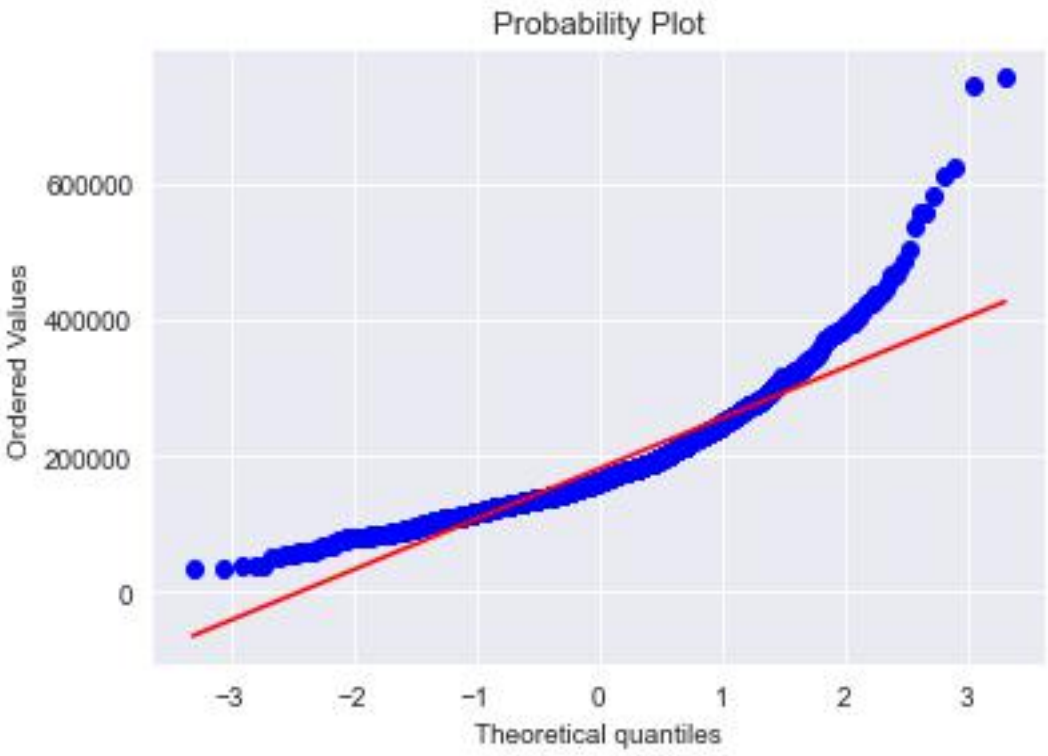


由上图可知，OverallQual(整体材料和完成质量)、GrLivArea(等级居住面积平方英尺)、GarageCars(车库停泊能力)、GarageArea(车库面积)、TotalBsmtSF(地下室总面积)、1stFlrSF(一楼平方英尺)、FullBath(完整浴室)、TotRmsAbvGrd(房间总级)、YearBuilt(建造年份)等几个特征和销售价格相关性比较高.

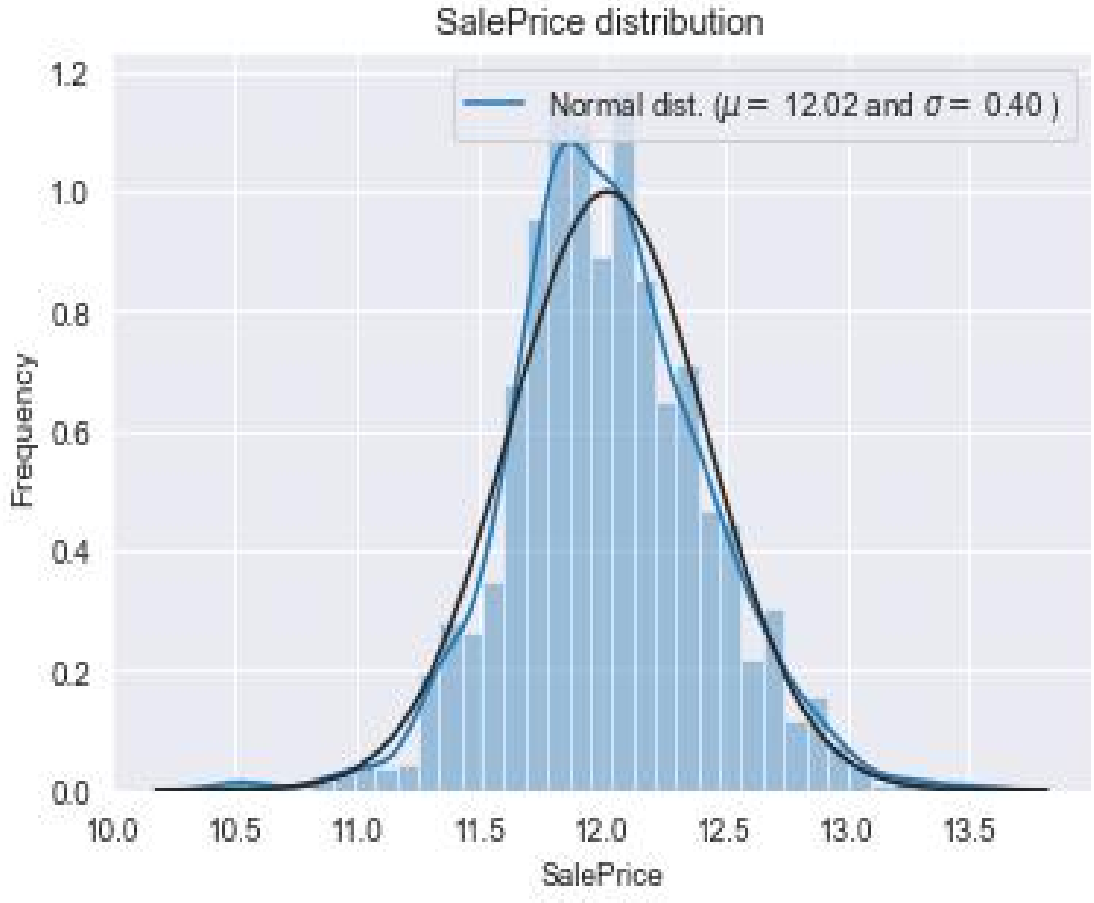
2.2.2 观察处理因变量 SalePrice

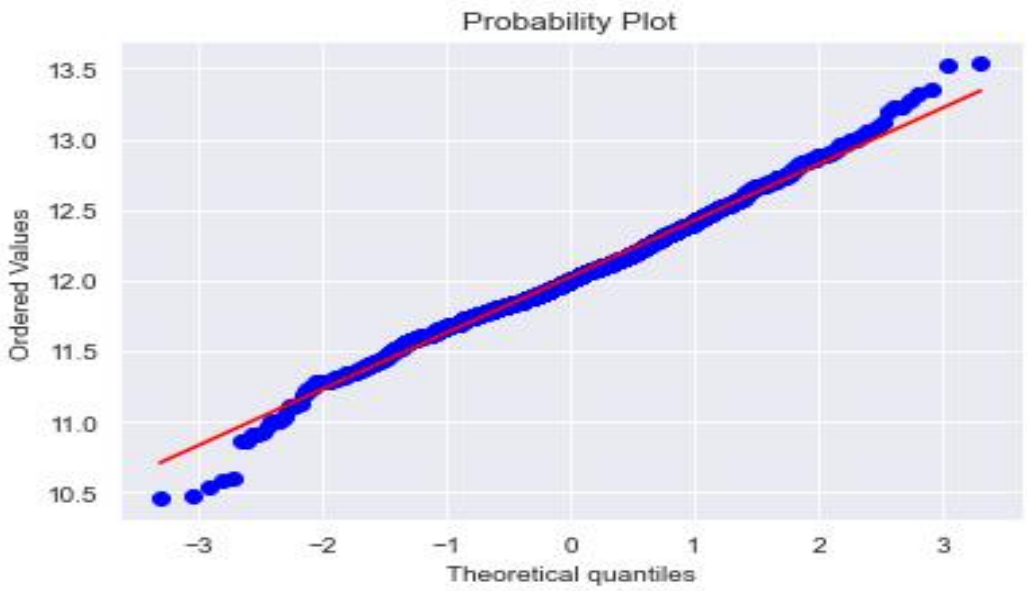
我们首先分析因变量 SalePrice，我们知道 SalePrice 作为我们的训练目标，只出现在训练集中，不出现在测试集，因此需要看看它长什么样子，也就是查看它的分布.





从结果来看并不满足标准的正态分布，目标变量向右倾斜. 由于(线性)模型喜欢正态分布的数据，为了我们分类器的学习更加准确，我们需要转换这个变量，使其更为正态分布.





2.2.3 特征值转换

我们注意到所有特征中 MSSubClass、OverallCond、YrSold、MoSold 其实是 category(类别) 的值

```
In [9]: print(all_data['MSSubClass'].dtypes)
        print(all_data['OverallCond'].dtypes)
        print(all_data['YrSold'].dtypes)
        print(all_data['MoSold'].dtypes)
```

```
int64
int64
int64
int64
```

但在数据集中却表现为数值类型的值，因此我们需要对这四个特征值进行类别转换，转换成 string 类型

```
In [10]: #MSSubClass=The building class
         all_data['MSSubClass'] = all_data['MSSubClass'].apply(str)


         #Changing OverallCond into a categorical variable
         all_data['OverallCond'] = all_data['OverallCond'].astype(str)


         #Year and month sold are transformed into categorical features.
         all_data['YrSold'] = all_data['YrSold'].astype(str)
         all_data['MoSold'] = all_data['MoSold'].astype(str)
```

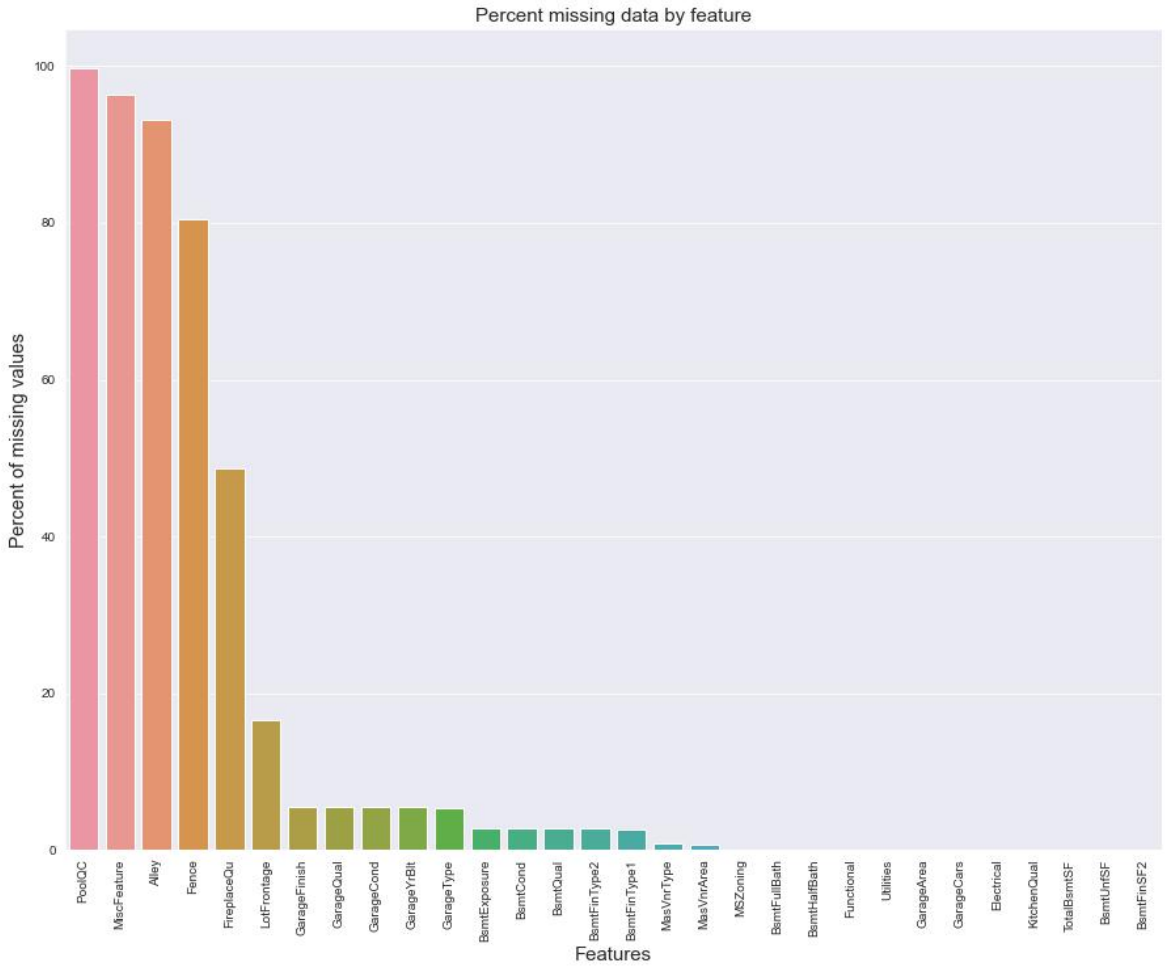
2.2.4 处理缺失值

在各个特征中，有不少是缺失部分值的，我们需要对这部分缺失的值进行处理. 首先我们打印缺失值. (all data 为连接 train 与 test 文件后的数据集)

Out[11]:

	Missing Ratio
PoolQC	99.691
MiscFeature	96.400
Alley	93.212
Fence	80.425
FireplaceQu	48.680
LotFrontage	16.661
GarageFinish	5.451
GarageQual	5.451
GarageCond	5.451
GarageYrBlt	5.451
GarageType	5.382
BsmtExposure	2.811
BsmtCond	2.811
BsmtQual	2.777
BsmtFinType2	2.743
BsmtFinType1	2.708
MasVnrType	0.823
MasVnrArea	0.788
MSZoning	0.137
BsmtFullBath	0.069

我们再把这些特征缺失数据的频度转化为直方图



接下来我们就需要对这些缺失值进行一些补充

• PoolQC：NA 表示“没有泳池”. 这是有道理的，因为丢失价值的比例很高(+99%)，而且大多数房子根本没有游泳池

```
all_data["PoolQC"] = all_data["PoolQC"].fillna("None")
```

• MiscFeature : NA 表示“没有混合特性”

```
all_data["MiscFeature"] = all_data["MiscFeature"].fillna("None")
```

• Alley : NA 表示“没有巷子通道”

```
all_data["Alley"] = all_data["Alley"].fillna("None")
```

• Fence : NA 表示“没有围栏”

```
all_data["Fence"] = all_data["Fence"].fillna("None")
```

• FireplaceQu : NA 表示“没有壁炉”

```
all_data["FireplaceQu"] = all_data["FireplaceQu"].fillna("None")
```

• LotFrontage : 由于与房产相连的每条街道的面积很可能与其附近的其他房屋面积相似，因此我们可以通过该街区的中间地块面积来填充缺失值.

```
all_data["LotFrontage"] = all_data.groupby("Neighborhood")["LotFrontage"].transform(
    lambda x: x.fillna(x.median()))
```

• GarageType, GarageFinish, GarageQual and GarageCond : 用“None”代替缺失值

```
for col in ('GarageType', 'GarageFinish', 'GarageQual', 'GarageCond'):
    all_data[col] = all_data[col].fillna('None')
```

• GarageYrBlt, GarageArea and GarageCars :将缺失数据替换为 0(因为没有车库=此类车库中没有汽车.)

```
for col in ('GarageYrBlt', 'GarageArea', 'GarageCars'):
    all_data[col] = all_data[col].fillna(0)
```

• BsmtFinSF1, BsmtFinSF2, BsmtUnfSF, TotalBsmtSF, BsmtFullBath and BsmtHalfBath : 由于没有地下室，缺失值值可能为零.

```
for col in ('BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF','TotalBsmtSF', 'BsmtFullBath', 'BsmtHalfBath'):
    all_data[col] = all_data[col].fillna(0)
```

• BsmtQual, BsmtCond, BsmtExposure, BsmtFinType1 and BsmtFinType2 :对于所有这些与地下室相关的分类特征, NaN 意味着没有地下室.

```
for col in ('BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2'):
    all_data[col] = all_data[col].fillna('None')
```

• MasVnrArea and MasVnrType : NA 很可能意味着这些房子没有砖石饰面. 我们可以为区域填充 0，为类型填充 None.

```
all_data["MasVnrType"] = all_data["MasVnrType"].fillna("None")
all_data["MasVnrArea"] = all_data["MasVnrArea"].fillna(0)
```

• MSZoning (一般分区分类)：“RL”是最常见的值. 所以我们可以用“RL”填充缺失的值.

```
all_data['MSZoning'] = all_data['MSZoning'].fillna(all_data['MSZoning'].mode()[0])
```

•Utilities：对于这个分类特征，所有记录都是“AllPub”，除了一个“NoSeWa”和两个 NA. 因为有“NoSewa”的房子在训练集中，这个特性对预测建模没有帮助. 然后我们可以安全地移除它.

```
all_data = all_data.drop(['Utilities'], axis=1)
```

• Functional : NA 意味着典型.

```
all_data["Functional"] = all_data["Functional"].fillna("Typ")
```

• Electrical：它有一个 NA 值. 由于这个特性主要是'SBrkr'，可以将缺失值设置为此.

```
all_data['Electrical'] = all_data['Electrical'].fillna(all_data['Electrical'].mode()[0])
```

•KitchenQual: 只有一个 NA 值，和 Electrical 一样，我们为 KitchenQual 中丢失的值设置“TA”(最常见).

```
all_data['KitchenQual'] = all_data['KitchenQual'].fillna(all_data['KitchenQual'].mode()[0])
```

• Exterior1st and Exterior2nd：同样，外部 1 和 2 只有一个缺失值. 我们将用最常用的字符串替换.

```
all_data['Exterior1st'] = all_data['Exterior1st'].fillna(all_data['Exterior1st'].mode()[0])
all_data['Exterior2nd'] = all_data['Exterior2nd'].fillna(all_data['Exterior2nd'].mode()[0])
```

• SaleType :用最频繁的 "WD"填充.

```
all_data['SaleType'] = all_data['SaleType'].fillna(all_data['SaleType'].mode()[0])
```

• MSSubClass : Na 很可能意味着没有建筑类. 我们可以用 None 替换缺失值.

```
all_data['MSSubClass'] = all_data['MSSubClass'].fillna("None")
```

同时，我们注意到区域相关特征对房价的决定非常重要，我们又增加了一个特征 TotalSF，即每套房子的地下室、一楼和二楼的总面积.

```
# Adding total sqfootage feature
all_data['TotalSF'] = all_data['TotalBsmtSF'] + all_data['1stFlrSF'] + all_data['2ndFlrSF']
```

3. 建模

3.1 分割数据集与交叉验证

通过第二部分的数据处理，我们已经把数据集中的特征基本上转化成了我们想要的数据集. 接下来就是建立模型训练我们的数据集. 一个模型能够重复已知的样本标签，但是不能预测任何有用的未知数据，这种现象称过度拟合. 为了避免过度拟合，我们采用的办法是，当做一个有监督的机器学习试验时，拿出一部分数据作为检验集 X_test, y_test. 在本项目里，我们定义一个交叉验证函数 rmsle_cv，同时函数中也包含了对数据集的随机分割.

In [23]:
```
#Validation function
n_folds = 5

def rmsle_cv(model):
    kf = KFold(n_folds, shuffle=True, random_state=42).get_n_splits(train.values)
    rmse= np.sqrt(-cross_val_score(model, train.values, y_train, scoring="neg_mean_squared_error", cv = kf))
    return(rmse)
```

3.2 模型建立

在本项目中，我们先导入 Lasso、 KernelRidge、 ElasticNet、 XGboost、 lightgbm 这几个模型，用上面定义的交叉验证函数来对这五个模型进行评分.

3.2.1 Lasso

```
score = rmsle_cv(lasso)
print("\nLasso score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))
#Lasso score: 0.1115 (0.0074)
```

Lasso(Least absolute shrinkage and selection operator)方法是以缩小变量集(降阶)为思想的压缩估计方法. 它通过构造一个惩罚函数，可以将变量的系数进行压缩并使某些回归系数变为 0，进而达到变量选择的目的.

lasso 回归的特色就是在建立广义线型模型的时候，这里广义线型模型包含一维连续因变量、多维连续因变量、非负次数因变量、二元离散因变量、多元离散因变，除此之外，无论因变量是连续的还是离散的，lasso 都能处理，总的来说，lasso 对于数据的要求是极其低的，所以应用程度较广；除此之外，lasso 还能够对变量进行筛选和对模型的复杂程度进行降低. 这里的变量筛选是指不把所有的变量都放入模型中进行拟合，而是有选择的把变量放入模型从而得到更好的性能参数. 复杂度调整是指通过一系列参数控制模型的复杂度，从而避免过度拟合(Overfitting).

3.2.2 KernelRidge

```
score = rmsle_cv(KRR)
print("Kernel Ridge score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))
#Kernel Ridge score: 0.1153 (0.0075)
```

Kernel Ridge Regression 即使用核技巧的岭回归(L2 正则线性回归).

3.2.3 ElasticNet

```
score = rmsle_cv(ENet)
print("ElasticNet score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))
#ElasticNet score: 0.1116 (0.0074)
```

ElasticNet 将 Lasso 和 Ridge 组成一个具有两种惩罚因素的单一模型：一个与 L1 范数成比例，另外一个与 L2 范数成比例. 使用这种方式方法所得到的模型就像纯粹的 Lasso 回归一样稀疏，但同时具有与岭回归提供的一样的正则化能力.

3.2.4 XGboost

```
score = rmsle_cv(xgb_model)
print("Xgboost score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))
#Xgboost score: 0.1164 (0.0070)
```

XGBoost 全名叫(eXtreme Gradient Boosting) 极端梯度提升，经常被用在一些比赛中，其效果显著. 它是大规模并行 boosted tree 的工具，它是目前最快最好的开源 boosted tree 工具包. XGBoost 所应用的算法就是 GBDT(gradient boosting decision tree)的改进，既可以用于分类也可以用于回归问题中.

3.2.5 lightgbm

```
score = rmsle_cv(lgb_model)
print("lightgbm score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))
#lightgbm score: 0.1288 (0.0058)
```

GBDT (Gradient Boosting Decision Tree) 是机器学习中一个长盛不衰的模型，其主要思想是利用弱分类器(决策树)迭代训练以得到最优模型，该模型具有训练效果好、不易过拟合等优点. GBDT 不仅在工业界应用广泛，通常被用于多分类、点击率预测、搜索排序等任务. 而 LightGBM(Light Gradient Boosting Machine) 是一个实现 GBDT 算法的框架，支持高效率的并行训练，并且具有更快的训练速度、更低的内存消耗、更好的准确率、支持分布式可以快速处理海量数据等优点.

3.3 模型集成

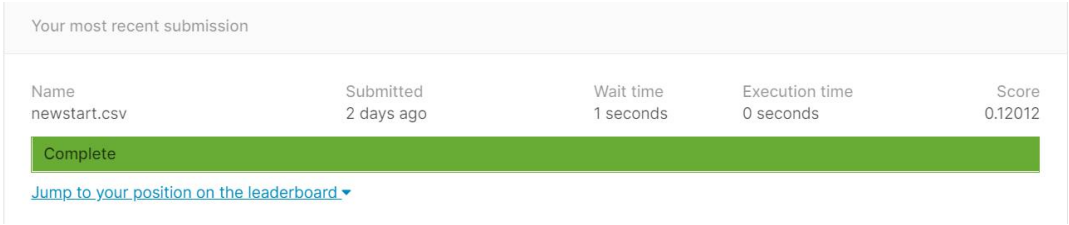
在模型集成这一部分，从平均基本模型的简单方法开始. 构建了一个新的类，用我们的模型扩展 scikit-learn. 使用的方法是最简单的叠加模型，在这里，我们平均了四个模型，ENet, GBoost, KRR 和 Lasso. 当然，也可以添加更多的模型.

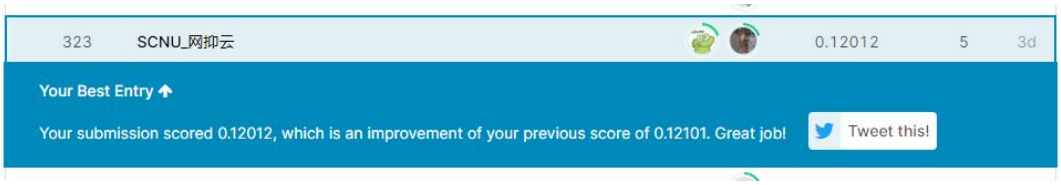
```
class AveragingModels(BaseEstimator, RegressorMixin, TransformerMixin):
    def __init__(self, models):
        self.models = models

    # we define clones of the original models to fit the data in
    def fit(self, X, y):
        self.models_ = [clone(x) for x in self.models]

        # Train cloned base models
        for model in self.models_:
            model.fit(X, y)

        return self

    #Now we do the predictions for cloned models and average them
    def predict(self, X):
        predictions = np.column_stack([
            model.predict(X) for model in self.models_
        ])
        return np.mean(predictions, axis=1)

averaged_models = AveragingModels(models = (ENet, GBoost, KRR, lasso))

score = rmsle_cv(averaged_models)
print(" Averaged base models score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))
#Averaged base models score: 0.1087 (0.0077)
```

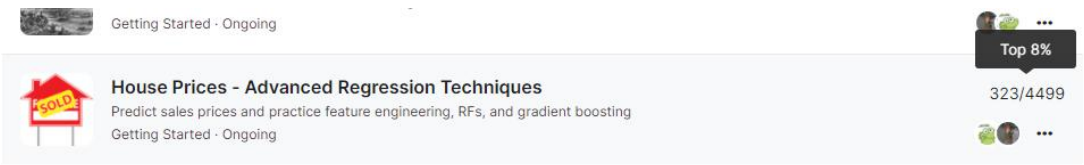
3.4 输出结果

```
submission = pd.DataFrame()
submission['Id'] = test_ID
submission['SalePrice'] = ensemble_result
submission.to_csv('newstart', index=False)
```

我们把 csv 结果上交到 Kaggle 官网







4. 总结

通过参加本次的 Kaggle 竞赛以及书写项目报告，基本了解了机器学习的流程，首先拿到一个数据集，需要分析哪个是目标值，哪些是有用的特征值，对无关的特征要进行降维处理．需要了解如何调用 sklearn 中的 API.

机器学习中，对于我们这些初学者来说，其实在算法方面只是会调 API，调整一下参数．所以数据的处理是机器学习中最重要的．数据处理不好，再优秀的算法也不能有效率的运行，更是对空间的损耗．所以我们再处理问题之前更多的是要考虑怎么把数据处理到最好，特征值怎么再分，缺失值怎么解决．

本次竞赛，我们使用了 Lasso, Ridge, ElasticNet, XGBoost 和 LightGBM 的集成模型对特征值再进行了处理，提高了测试的准确率． 基本入门机器学习，但还是需要后面紧跟课程的学习，将机器学习中学到的知识灵活运用，以便准备后面深度学习的大项目．

参考文献

[1] https://blog.csdn.net/weixin_48994268/article/details/109686341
[2]https://blog.csdn.net/weixin_36194102/article/details/113016852
[3]https://blog.csdn.net/u012063773/article/details/79349256